# Experiment 5

**Student Name:** Tarun Baliyan **UID:** 22BET10184

**Branch:** IT **Section/Group:** 22BET\_IOT-702/A

**Semester:** 6th **Date of Performance:** 26/02/25

**Subject Name:** Project Based Learning Subject in Java with Lab

**Subject Code:** 22ITH-359

**Problem 1: Develop a Java application using ArrayList to efficiently manage employee records by allowing users to add, update, remove, search, and display employee details.**

1. **Aim:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).
2. **Objective:**

Demonstrate autoboxing and unboxing in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

1. **Implementation/Code:**

# import java.util.ArrayList;

# import java.util.List;

# public class AutoboxingExample {

# public static void main(String[] args) {

# String[] numberStrings = {"10", "20", "30", "40", "50"};

# List<Integer> numbers = parseStringArrayToIntegers(numberStrings);

# int sum = calculateSum(numbers);

# System.out.println("The sum of the numbers is: " + sum);}

# public static List<Integer> parseStringArrayToIntegers(String[] strings) {

# List<Integer> integerList = new ArrayList<>();

# for (String str : strings) {

# integerList.add(Integer.parseInt(str)); }

# return integerList; }

# public static int calculateSum(List<Integer> numbers) {

# int sum = 0;

# for (Integer num : numbers) {

# sum += num;}

# return sum;}}

# 4. Output:

## 



# 5. Learning Outcomes:

• Understand the concept of autoboxing and unboxing in Java and how primitive types are

automatically converted to their wrapper classes and vice versa.

• Learn how to convert string values into Integer objects using Integer.parseInt() and store

them in a list.

• Gain experience in working with ArrayLists to store and manipulate a collection of

numbers dynamically.

• Develop proficiency in iterating through collections and performing arithmetic

operations like summation.

**Problem 2: Develop a Java program to serialize and deserialize a Student object, preserving its state for storage and retrieval.**

1. **Aim:** Create a Java program to serialize and deserialize a Student object
2. **Objective:**  The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.
3. **Implementation/Code:**

import java.io.\*;

// Student class implementing Serializable

class Student implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

private double gpa;

public Student(int id, String name, double gpa) {

this.id = id;

this.name = name;

this.gpa = gpa;

}

@Override

public String toString() {

return "Student{id=" + id + ", name='" + name + "', gpa=" + gpa + "}";

}

}

// StudentSerialization class to handle serialization and deserialization

public class StudentSerialization {

private static final String FILE\_NAME = "student.ser";

public static void main(String[] args) {

Student student = new Student(1, "Anwar", 7.8);

serializeStudent(student);

deserializeStudent();

}

public static void serializeStudent(Student student) {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_NAME))) {

oos.writeObject(student);

System.out.println("Student object serialized successfully.");

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

} catch (IOException e) {

System.err.println("IOException occurred: " + e.getMessage());

}

}

public static void deserializeStudent() {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE\_NAME))) {

Student student = (Student) ois.readObject();

System.out.println("Deserialized Student: " + student);

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

} catch (IOException e) {

System.err.println("IOException occurred: " + e.getMessage());

} catch (ClassNotFoundException e) {

System.err.println("Class not found: " + e.getMessage());

}

}

}

# 4. Output:

## 



# 5. Learning Outcomes:

• Understand object serialization and deserialization in Java.

• Learn how to use ObjectOutputStream and ObjectInputStream for file operations.

• Implement exception handling for FileNotFoundException, IOException, and

ClassNotFoundException.

• Gain hands-on experience in storing and retrieving objects from a file.

**Problem 3: Develop a Java application to manage employee records efficiently by adding, storing, and displaying employee details using a menu-driven approach.**

1. **Aim:** To create a file-based menu-driven Java application for adding and retrieving employee details.
2. **Objective:**

 Implement a menu with options to add, display, and exit.

 Store employee details (name, ID, designation, salary) in a file.

 Retrieve and display stored employee records.

 Ensure data persistence and proper error handling.

1. **Implementation/Code:**

import java.io.\*;

import java.util.\*;

// Employee class implementing Serializable for persistence

class Employee implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

private String designation;

private double salary;

public Employee(int id, String name, String designation, double salary) {

this.id = id;

this.name = name;

this.designation = designation;

this.salary = salary;

}

@Override

public String toString() {

return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary;

}

}

// Employee Management System with serialization and deserialization

public class EmployeeManagementSystem {

private static final String FILE\_NAME = "employees.ser";

private static List<Employee> employees = new ArrayList<>();

public static void addEmployee() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Employee ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter Employee Name: ");

String name = scanner.nextLine();

System.out.print("Enter Designation: ");

String designation = scanner.nextLine();

System.out.print("Enter Salary: ");

double salary = scanner.nextDouble();

Employee employee = new Employee(id, name, designation, salary);

employees.add(employee);

saveEmployees();

System.out.println("Employee added successfully!");

}

public static void displayAllEmployees() {

loadEmployees();

if (employees.isEmpty()) {

System.out.println("No employees found.");

} else {

for (Employee employee : employees) {

System.out.println(employee);

}

}

}

private static void saveEmployees() {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_NAME))) {

oos.writeObject(employees);

} catch (IOException e) {

System.err.println("Error saving employees: " + e.getMessage());

}

}

@SuppressWarnings("unchecked")

private static void loadEmployees() {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE\_NAME))) {

employees = (List<Employee>) ois.readObject();

} catch (FileNotFoundException e) {

employees = new ArrayList<>();

} catch (IOException | ClassNotFoundException e) {

System.err.println("Error loading employees: " + e.getMessage());

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nEmployee Management System");

System.out.println("1. Add an Employee");

System.out.println("2. Display All Employees");

System.out.println("3. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1 -> addEmployee();

case 2 -> displayAllEmployees();

case 3 -> {

System.out.println("Exiting...");

return;

}

default -> System.out.println("Invalid choice! Please try again.");

}

}

}

}

# 4. Output:

## 



# 5. Learning Outcomes:

* Understand file handling and serialization in Java to store and retrieve objects persistently.
* Learn how to implement a menu-driven console application using loops and conditional

statements.

* Gain experience in object-oriented programming (OOP) by defining and managing

Employee objects.

* Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
* Develop skills in list manipulation and user input handling using ArrayList and Scanner.